Programs High to Low

At the lowest level, the operation of the computer is the numerical manipulation of the memory.

A **program** is a sequence of instructions, ultimately expressed in **machine language** and understood by the specific **CPU** (Central Processing Unit, often just called the **processor**). Like all data, a program is represented digitally, as bytes, that are loaded into the memory to be executed.

The basic operations of a processor include reading and writing values to the memory, basic math operations (adding, multiplication), and **conditionals**, allowing the flow of a program (its step by step execution) to take different paths depending on a value in memory. **Loops and branches** are both examples of this capability of the processor. The computer's "interactivity" is in one sense the ability for the path of a program (or even the instructions themselves) to change in reaction to data in the memory, including user input.

In order for programs to work with the available memory, every portion of memory can be referred to by a numeric **address**. Programming languages allow the program to create names for memory locations, these named memory locations are called **variables.**

```
        MOVEQ.L  #99, D0
        LEA      P, A0
        LEA      Q, A1
        CLR.L    D1
LOOP:   MOVE     (A0)+, D2
        MULS     (A1)+, D2
        ADD.L    D2, D1
        DBF      D0, LOOP
```

Assembly language of a FOR loop for a Motorola 68000 processor (used in the first Mac)
When **compiled**, each line translates into binary machine language that the processor is designed to process.

High-level languages are more abstracted from the underlying processor instructions. One instruction in a high level language may translate into a large number of machine language instructions. Compilers are programs that translate code written in progamming languages to the binary instructions a specific processor can work with.

Python is an example of a high-level language. **C** is relative to Python a lower-level language in that many of the details of manipulating the memory are left "exposed" to the programmer.

Higher level languages provide built-in mechanisms (routines or procedures) for the tasks commonly associated with every program (managing memory, creating basic **data structures** like lists and dictionaries, searching and sorting data). In this way, programmers can work at a "higher level" and not need to rewrite the same code over and over.

However, it is important to understand how lower level code works, especially when trying to **optimize** code (get the best performance out of code), or sometimes to be able to understand what's happening when **debugging** (the process of testing and looking for errors in a program).